SUPPLEMENT TO LABORATORY 1

Instructional Format

The instructional line has four "fields", shown below.

The first field is that of the address. It can be a string of alphanumeric characters, including underlining, but it must begin with an alpha character. The length is "assembler" dependent as well as the delimiter shown, the colon. It is recommended that the colon be used since most assemblers presently seem to accept it as an option, if not a requirement.

The second field is for the mnemonic operation of the instruction. The delimiter between it and the third field used by an operand, is a space. Not all instructions require an operand and thus, the field may not be used.

The fourth field as a semicolon as a delimiter and is reserved for comments.

ADDRESS: OPERATION OPERAND; COMMENT

Operations and Operands

Operations are "actions" that the computer is to perform. The actions take place even when the computer has no operand. For example, the instruction, NOP, is explicitly an instruction that is the mnemonic for "no operation", and yet an operation does take place: cycle times are executed by the computer as a result of the execution of this instruction, which is often its purpose.

Operands are the things that are necessary for the operation to take place. They may be explicitly stated, implicitly indicated or both. For example, the instruction,

Idaa #$3b

"Load (Id) the Accumulator (a) A register (a) with the number (#) that is in hexadecimal format ($) and whose value is 3b", has both implicit and explicit operands.
The explicit operand is the number, $3b, delineated from the operation, ldaa by a space, while the implicit operand is the register receiving the action, accumulator A. However, A is not explicitly stated and does not appear in the operand field, but is "buried" in the operation mnemonic, ldaa.

In general, operands will appear as:

 Registers
 Registers whose contents are offset by numbers
 Addresses, either numerical or as a string of characters
 Numbers, in binary (%), hexadecimal ($) or decimal (!, @, &)

The Operations Most Commonly Used

There are less than thirty instruction mnemonics that are needed for all laboratories. A list of these is:

 Loading and Storing (Explicit operands are numbers: # followed by formatted numbers or strings; or formatted numbers or strings which correspond to memory locations.)

 ldaa staa
 ldab stab
 ldx stx
 ldy sty
 ldd std

 Incrementing (Operands are all implicit. Can also use alternate form, "decrementing")

 inca
 incb
 inx
 iny

 Branching (Operands are explicit. There are several other flag forms)

 bra
 bne
 bsr
 jsr
 rts
 rti

 Comparing ((Explicit operands are numbers or memory locations. Several other forms with different operands)
cmpa

**Arithmetical** (Explicit operands are numbers or memory locations.)

adda
addd

**Logical** (Explicit operands are numbers or memory locations.)

ora
anda

**Special**

cli
wai

**Directives**

Directives are instructions to the "loader" that requested by the programmer through the "assembler" and are not operations to be performed by the computer. A list of directives that are used to set aside memory for the storage and initialization of numerical data, grouped according to a common action and recognized by some assemblers, is given in the following table.

<table>
<thead>
<tr>
<th>DIRECTIVE</th>
<th>AFFECT AT MEMORY LOCATION IN PROGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>db N</td>
<td>Define the value of a byte (8-bits) whose value is N</td>
</tr>
<tr>
<td>dc.b</td>
<td>Define a constant byte</td>
</tr>
<tr>
<td>fc.b</td>
<td></td>
</tr>
<tr>
<td>dw N</td>
<td>Define the value of a word (16-bits) whose value is N</td>
</tr>
<tr>
<td>dc.w</td>
<td></td>
</tr>
<tr>
<td>fdb</td>
<td></td>
</tr>
<tr>
<td>ffc</td>
<td></td>
</tr>
<tr>
<td>rmb</td>
<td></td>
</tr>
</tbody>
</table>

You might use the program below to answer the questions that follow:
org $1000
ldx  #data  ;Load THE NUMBER (#) that is “data” into X.
ldaa data  ;Load the CONTENTS of address “data” into A.
staa data+13 ;Store the contents of A into the location “store”, 13
              ;Locations past “data”.
here:  bra here  ;Loop here to keep machine going.

org $2000
data:  db $2c,$b4,%10110110,"EE 101/134"
store: ds !10  ;Reserve 10 locations of memory whose contents are
              ;initially unknown.

1. A "breakpoint has been set and execution stops. What is printed and in what
   order?
2. How do you make the program continue execution after a breakpoint?
3. What are two ways, each using a single command, to execute the first three
   instructions of a program?
4. What single command will allow you to set up to change the contents of the X
   register?